Phenix Timing System

Stephen Adler[a]

[a] Brookhaven National Laboratory

Abstract

This technical note describes how to configure a Granule Timing Module (GTM) using a configuration text file.

1 Generating raw mode bit data file

A *mode bit language* has been developed in order to ease the task of generating mode bit data files which are used to load into the Grandual Timing Modules. The idea is to write an ascii text file using this mode bit language syntax, run a program called MBParce which will then translate the *mode bit source code* into a raw mode bit file. The GTM's will then load the data in this raw mode bit file into the various memory banks and registers of the grandual timing module.

So one needs to learn this mode bit lanuage in order to "program" the GTM. This is a very simple, but rather cryptic language which one often needs to resort to an example file to remember how it works. Further, in order to understand it, and why its cryptic form, one needs to understand how a GTM works and its internals.

2 How a GTM works

A GTM is a rather complicated pulse generator. Its designed to provide timing pulse sequences to virtually any device you can think of which will operate in the RHIC accelerator environment. What this means is that the RHIC operates by injecting 120 bunches of particles into each of the two counter rotating rings which make up the RHIC accelerator. These two sets of 120 bunches will then generate 120 possible times during one circulation period, that a collision may occure. Thus, the GTM has been designed such that one can generate pulse sequences in sets of 120 cycles, which are repeated over and over again, to match the particular beam conditions for each one of the 120 bunches in the ring.

This leads one into the following design philosophy of the GTM. The GTM is made up of two memory sections. The first one is called the schedualar mode bit memory, and the second one is called the schedualr command memory. What one does is load the mode bit sequences into the mode bit memory, and the number of times each one of these sequences is to repeat into the command memory. One could think of the memories as being data memory and program memory. The GTM program is loaded into the command memory (program memory) and the mode bits them selves, (the mode bit data), gets loaded into the mode bit memory (the data memory.) Finally, the GTM has a sequencer. (i.e. a CPU) What this sequencer does is to read memory location 0 of the command memory, (which contains the number of time to loop over one of the mode bit patterns) and "execute" that command which means to loop over the mode bits N number of times. It then reads the next memory location in the command memory and executes that command. This process repeats until the sequencer executes a command which includes a reset. This means that after looping through the mode bit pattern N times, it will go back to memory location 0 and start the process over again. Thus the GTM will loop over the commands in the command memory, which will cause the sequencer to loop over the mode bits. So the GTM loops and loops and loops over a huge phase space of mode bit sequences.

The syntax of the mode bit language is designed to mimick this looping processes. Basically their are two sections to a mode bit "program". The first one details the data

which will be loaded into the mode bit memory, and the second one are the commands which are loaded into the command memory.

2.1 Mode bit data section

The mode bit data sections is defined by the following syntax:

```
LABEL: label
FEM:
mode bit data sequence
LL1:
mode bit data sequence
PPG:
mode bit data sequence
DUMP:
```

The text in bold is required by the syntax.
The *mode bit data sequence* is a string which details the mode bit data which will be generated for one 120 count loop. The syntax of this string is as follows:
`n[;m][xo][,]`
where *n, m*, are hex numbers and *o* is a decimal number. A string of hex numbers seperated by the ";" character indicates a repeat sequence. A string of hex numbers seperated by "," is just a regular sequence. A repeat sequence or a hex number can be repated o by use of the "x" operator. The following is a list of examples which will clarify the syntax.

Example1:
`00x120`
Expands to 120 values of 0.

Example2:
`10x60,20x60`
Expands to 60 values of hex 10, and 60 values of hex 20.

Example3:
`1,2,3,4,5,6x115`
Expands to 1, 2, 3, 4, 5 followed by 6 repeated 115 times.

Example4:
`1;2x60`
Expands to 60 repeated sequences of 1, 2. (i.e. 1,2,1,2,1,2,...,1,2).

The string after the `LABEL:` tag identifies the mode bit command section. This string is called the *mode bit command tag*. One can include as many different mode bit command sections in the "GTM program" but each is identified by a unique *mode bit command* tag.

The labels of `FEM:`, `LL1:` and `PPG:` preceeding a *mode bit data sequence* string indicates to which system the mode bit data sequence is for. `FEM:` indicates that the mode bits will be delivered to the Front End Module. `LL1:` indicates that the mode

bits specified by the mode bit data sequence will go to the Local Level 1 system, and `PPG:` means that the mode bits are destined for the programmable pulse generator.

Finally, the `DUMP:` string terminates the mode bit data sequence block. You can start another mode bit data sequence by using the `LABEL:` tag or begin the command section block which is described next.

2.2 Mode bit command section

The mode bit command section is defined by the following syntax: **`START:`**
`sequencer command`
`sequencer command`
`sequencer command`
The text in bold is required. This means that you need to have at least one sequencer command line. The sytax for the sequencer command line is:
`LABEL[xI][;break][;reset]`

Where `LABEL` is a *mode bit command* tag, i is a hex number between 1 and 0x800000. The string `;break` indicates the end of the initialization sequence, and the string `;reset` indicates the end of the continous sequence. The idea being that several mode bit command blocks are executed once after the GTM sequencer is started, and then a second set of mode bit command blocks is executed in order indefinatly. The xI string indicates that the mode bit command block is to be executed I times. The following set of examples will clarify the syntax.

> Example: 3 mode bit command blocks have been defined labeled A, B and C.
>
> ```
> START:
> A;BREAK
> Bx10
> Cx20;RESET
> ```
>
> The above sequencer command section will direct the sequencer to execute the `A` mode bit command block once, execute the `B` mode bit command block hex 0x10 times, execute the `C` mode bit command block hex 0x20 times, then go back to the line after the `;BREAK` statement.
>
> Example: 4 mode bit command blocks have been defined labeled M, N, and O.
>
> ```
> START:
> Mx10
> Nx20;BREAK
> O;RESET
> ```

The above sequencer command block will direct the sequencer to execute the `M` mode bit command block hex 0x10 times, then execute the `N` mode bit command block hex 0x20 times, and execute the `O` command block an indefinate number of times. (i.e. the sequencer will loop indefinatly executing the O mode bit command block.)

This concludes the documentation for the GTM mode bit language. What follows are a couple of complete examples of mode bit programs.

Example: The following mode bit program is written for the Drift Chamber FEM. It executes the mode bit reset command once (0x02) and then executes the run mode bit command indefinatly.

```
LABEL: A
FEM:
00x119,02
DUMP:

LABEL: B
FEM:
10x120
DUMP:

START:
A;BREAK
B;RESET
```

Example: The following mode bit program is used by the Pad Chamber to execute the test strobing sequence. The mode bit commands of 0xF7 and 0xF4 will force an internal strobe of a cirtain number of Pad Chamber ROC's. 51 clock cycles later the PPG issues a TTL pulse which is feed into the GTM forcing a level accept into the PC FEM which causes it to readout the "internally strobed" data.

```
LABEL: A  # Mode bit data block which issues a strobe followed by
          # level one accept issued by the PPG.
FEM:
10x4,12x2,10x18,f7x10,f4x10,10x76
PPG:
00x95,01,00x24  # toggle bit 1 of ppg 51 clock ticks after the Strobe.
DUMP:           # dump out the mode bits

LABEL: B        # Mode bit data block B
FEM:
10x120          # 120 values of 0x10, the run mode bit command.
DUMP:           #dump out the mode bits.
```

```
START:          # Start of schedular code
A               # execute mode bits labled A once
Bx400;RESET     # execute mode bits labled B 0x400 times, this delays
                # the next strobe for 0x400 times 120x10Mhz clock.
                # (i.e. about 12 ms or 81 Hz)
```

3 GTM raw data file format

For completness sake, the GTM raw data file format will be described. The format of this file follows closley the address space of the GTM which can be viewed in Figure 1.

The GTM raw data file is an ascii file composed of a list of hex numbers. See Figure 2 and ignore the text to the right of the 8 digit hex numbers since they are there to help decode the file format. The first number in the file is the number of words to follow (XWC) which are to be written into the Scheduler mode bit address space. The Schedular mode bit data should follow, again a sequence of ascii hex numbers seperated by a line feed. Next is the number of words to be loaded into the Schedualr command address spaced, followed by the data to be loaded into this space. Finally, 8 words terminate the file which are loaded into six consecutive registers in the register address space plus two more bit fields in other parts of the GTM register space. The definition of these registers can be found in table 1. The example in Figure 2 is technically correct, but some what meaning less since technically, one should have at least 128 words of schedular mode bit data to write into the first block of mode bit data memory. But putting 128 words into the example in Figure 2 would make the figure itself rather ugly.

GTM Auxillary Register #	Description
1	FEM Count N Dead Time, Calculated by MBParce
2	FEM Convert Time
3	FEM Enable Data Time
4	GTM Clock find delay adjust
5	GTM clock course delay adjust
6	GTM rough delay adjust
7	GTM Count N Dead Time
8	GTM Level 1 Delay Clock Count

Table 1: Definition of GTM loadable register space

Finally, what follows is a set of sample Mode bit program files along with the generated output which is read in by the GTM software.

Filename	Description
GTM.DC.E	Sample DC FEM mode bit file.
GTM.DC.E.gtm	Output of MBParce on the above file
MBParce.cc	Source to MBParce

Figure 1: GTM Address space.

```
00000008  <-- XWC
00000000  ------
00000001    |
00000002    D
00000003    A
00000004    T
00000005    A
00000006    |
00000007  ------
00000001 <-- XWC
00000001 <-- Data
00000010  ------
00000011    |
00000012    |
00000013   reg
00000014   data
00000015    |
00000016    |
00000017  ------
```

Figure 2: Sample raw data file. See text for further details.

3.1 MBParce details

The following is how you build the program:
Sun

```
cc -o MBParce -D__SunOS_5_6 MBParce.c}
```

Linux and Irix

```
cc -o MBParce MBParce.c
```

To execute the command and generate a raw GTM data file, type the following.

```
./MBParce GTMProgramFileName > GTMRawDataFilename
```

Note, if you are using an old version of the GTM hardware, then you need to add a '-old' qualifier to the command as such:

```
./MBParce -old GTMProgramFileName > GTMRawDataFilename
```

Contact Steve Adler in case you are unsure which version of the GTM you have.

3.2 The vxWorks GTM software

To load the software in vxWorks, type at the vxWorks prompt shell:

```
ld < libGTMTools.o
```

To run the gtm software type gtm at the vxWorks prompt. You will be presented with a menu of commands which are slected by typing in the menu item number at the prompt.

4 Transcriber's note

Steve Adler's original documentation on the GTM was posted at:
`http://ssadler.phy.bnl.gov/adler/phenix/timing/TimingSystem.html`
which was taken down some time after his departure.

A copy of the page was posted by John Haggerty at:
`http://www.phenix.bnl.gov/phenix/project_info/electronics/timing/ssadler/TimingSystem.html`
which was then used to generate this document.

-John Koster